# Robot Motion Planning on a Chip

Sean Murray, Will Floyd-Jones[*], Ying Qi[*], Daniel Sorin and George Konidaris
Duke Robotics
Departments of Computer Science and Electrical & Computer Engineering
Duke University, Durham NC 27708

***Abstract*—We describe a process that constructs robot-specific circuitry for motion planning, capable of generating motion plans approximately three orders of magnitude faster than existing methods. Our method is based on building collision detection circuits for a probabilistic roadmap. Collision detection for the roadmap edges is completely parallelized, so that the time to determine which edges are in collision is independent of the number of edges. We demonstrate planning using a 6-degree-of-freedom robot arm in less than $1$ millisecond.**

## I. Introduction

Recent advances in robotics have the potential to dramatically change the way we live. Robots are being developed for a wide spectrum of real-world applications including health care, personal assistance, general-purpose manufacturing, search-and-rescue, and military operations. Unlike traditional robotics applications in which a robot operates in a tightly controlled environment (e.g., on an assembly line where the environment is static and known a priori), these applications require the robot to adapt to its environment at runtime, which means that it must perform motion planning [15].

Motion planning is the problem of determining how a robot should move to achieve a goal; for example, a robot may wish to move its arm to reach a desired object without colliding with itself or any other object. It is a heavily studied algorithmic problem with several well-known solutions. Unfortunately, existing motion planning algorithms running on current hardware are generally too slow to plan in real-time for complex robots and environments. The fastest existing implementations run on graphics processing units (GPUs) [3, 17, 23, 24, 25, 26], requiring hundreds of milliseconds to run and consuming substantial power, which might be infeasible for untethered robots or large-scale factory settings. The ability to perform motion planning is critical to dealing with natural environments that are not carefully controlled or designed, and our inability to generate plans in real-time is a major obstacle preventing the widespread deployment of robots in the workplace and the home.

We propose the use of *specialized hardware* for robot motion planning—specifically, we describe a process that generates robot-specific circuitry that exploits a combination of aggressive precomputation and massive parallelism to generate motion plans for real robots approximately three orders of magnitude faster than existing GPU implementations. We implement our circuits on a field-programmable gate array (FPGA), which is able to find plans for the Jaco-2 6-degree-of-freedom arm in less than 1 millisecond[†].

[*]Equal contribution.
[†]A video showing the Jaco arm and our chip in action can be seen at: https://youtu.be/u4snHh_S_Ao

## II. Background

We can describe the pose of an articulated robot—one consisting of a set of rigid bodies connected by joints—by assigning positions to all of its joints. The space of all such joint positions is called the robot's configuration space (often called C-space) [19], denoted $Q$. Some configurations would result in a collision with an obstacle in the robot's environment, a description of which is assumed to be given before planning commences, or the robot's own body; the remainder of $Q$ is called *free space*. The robot motion-planning problem consists of finding a path through free space from some start configuration $q_0$ to any configuration within a target set $G$.

A common approach to solving motion planning problems is to build a probabilistic roadmap [13], or PRM. A PRM consists of a graph where each node is a point in free-space, and a pair of points can be connected if a straight-line movement between them is possible without a collision. This is depicted in Figure 1. Given a $(q_0, G)$ pair, we can find a motion plan by sampling nodes in free space, attempting to find a collision-free connection from each node to its nearest neighbors, and then eventually finding a path through the graph from $q_0$ to some node in $G$. If the nodes in the PRM are sampled randomly from C-space, the PRM is probabilistically complete—guaranteed to find a solution if one exists, with

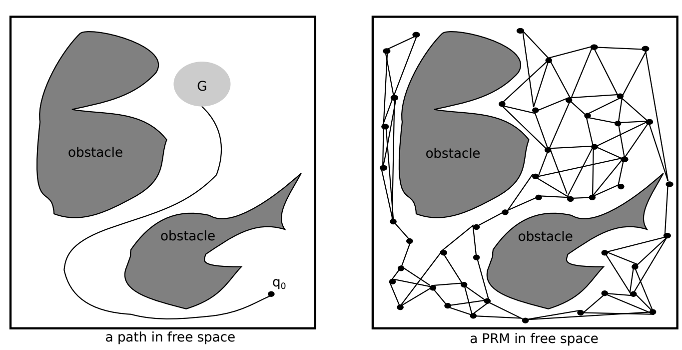



Fig. 1: The goal of motion planning is to find a path in free space (left). A PRM is a graph consisting of points in free-space, with edges connecting points when a direct movement between them is collision-free (right).

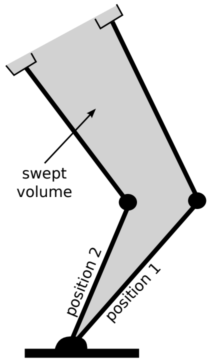



Fig. 2: A swept volume refers to the space covered by a movement between two robot configurations (equivalently, two points in C-space). The dominating computational expense in sample-based motion planning algorithms is determining whether a swept volume overlaps with an obstacle.

probability 1 in the limit of the number of samples.

In practice, the dominating computational expense in constructing a PRM is determining whether a straight-line path between two points in configuration space (i.e., directly moving each joint from one position to another) collides with an obstacle; one study [3] showed that collision detection consumed $99\%$ of the compute time of a sample-based motion planner. There are techniques to shift the asymptotic bottleneck from collision detection to nearest-neighbor search by memoizing certain information while collision checking and amortizing the cost over a large number of nodes [4]. However, the constants attached to these terms are such that the roadmap must be extremely dense for the computational burden of nearest neighbor search to approach that of collision detection.

Collision detection requires us to build a geometric model of the swept volume of each robot movement—we illustrate a volume swept by an example movement in Figure 2—and to test this model against a geometric model of each obstacle. The models are typically CAD-like mesh models, since robot arms and real obstacles often have irregular or complex shapes.

The PRM is a multi-query algorithm: a great deal of computational effort must be expended to construct the graph, but it can then be reused to find plans for new $(q_0, G)$ pairs. However, direct reuse is only possible if the obstacles are unchanged. When the obstacles change, we must re-compute the connectivity [18], which is quite expensive. There has been work on re-planning algorithms that attempt to leverage spatial knowledge to reduce the burden of dealing with environment changes [2, 22], but these still involve significant computation. Consequently, most planning algorithms are single query, and are rerun from scratch for each $(q_0, G)$ pair (only including edges on the shortest path from $q_0$ to some other node, resulting in a tree rather than a graph [16, 11]). However, even these algorithms are insufficiently fast to perform real-time planning on a commodity CPU for realistically complex robot bodies and real environments.

Several research groups have sought to exploit the performance of GPUs to improve the performance of collision detection algorithms [3, 8, 14, 20, 23, 24, 25, 26], for both motion planning and for use in graphics and simulations. This technique uses the many-core architecture of GPUs to exploit more parallelism from planning algorithms than a CPU can extract. This area of research has demonstrated that with some amount of algorithm tuning, collision detection can be performed at substantially higher performance on GPUs than on CPUs. Nevertheless, the performance and performance/watt are still insufficient for real-time motion planning for complex robots in real environments.

## III. Robot Motion Planning on a Chip

Our strategy broadly follows Leven and Hutchinson [18]: we first build a single general-purpose PRM—in advance—by assuming that the robot exists in an environment with no obstacles. We then adapt the PRM to a particular problem instance by removing the edges that collide with obstacles of that instance. Finally, we create a motion plan by simply finding a path through the resulting (smaller) PRM.

We use the following process, as depicted in Figure 3:

**Once, At Design Time**

1) Given a robot description, we produce a single PRM, only checking for collisions with permanent features and self-collisions. Since we will be reusing this PRM, it is important that it has a high probability of containing a plan for any particular scenario the robot may find itself in. Therefore, we typically construct a very large PRM and subsample it for coverage; we discuss this step in the following section. (Figures 3a and b.)
2) Each edge in the PRM corresponds to a swept volume. We discretize the 3D space around the robot into depth pixels, and obtain a list of the depth pixels that are in collision with each edge's swept volume. Note that the discretization need not be uniform—for example, we might discretize more finely in areas close to the robot, or where finer movements are likely to be required (Figure 3c).
3) We encode depth pixel coordinates in binary, and construct a logical expression for each edge that returns `true` if a given depth pixel is in the collision list for that edge, and `false` if not. (Figure 3d.)
4) We construct and optimize a collision detection circuit (CDC) for each edge using its logical expression. Each CDC takes as input a binary description of a depth pixel and outputs a single bit, indicating whether or not the input depth pixel causes the edge to be in collision. Since an edge is in collision if it collides with *any* depth pixel, the results of *all* pixel collision evaluations are ORed together, and the resulting collision bit is stored. Our processor consists of the CDC circuitry for every edge. (Figure 3e.)

**At Runtime**

5) When a robot wishes to solve a motion planning problem, it resets the collision bits for each edge. It then senses

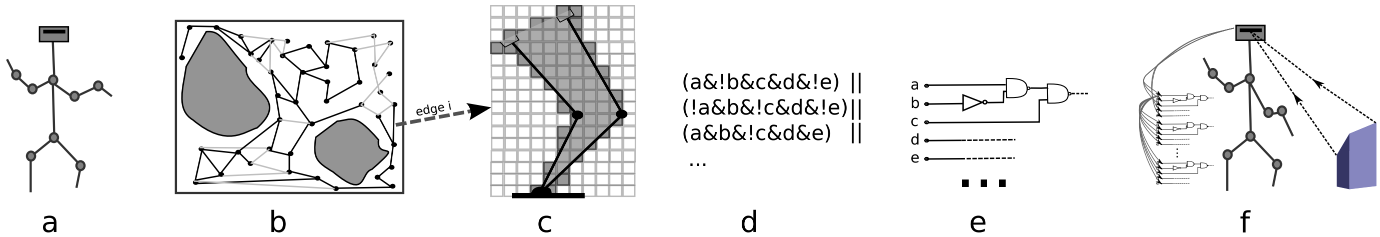



Fig. 3: Our process for producing robot-specific motion planning circuitry. Given a robot description (a), we construct a PRM (b), most likely subsampled for coverage from a much larger PRM. We discretize the robot's reachable space into depth pixels and, for each edge $i$ on the PRM, precompute all the depth pixels that collide with the corresponding swept volume (c). We use these values to construct a logical expression that, given the coordinates of a depth pixel encoded in binary, returns `true` if that depth pixel collides with edge $i$ (d); this logical expression is optimized and used to build a collision detection circuit (CDC) (e). For each edge in the PRM there is one such circuit. When the robot wishes to construct a motion plan, it perceives its environment, determines which depth pixels correspond to obstacles, and transmits their binary representations to every CDC (f). All CDCs perform collision detection *simultaneously, in parallel* for each depth pixel, storing a bit which indicates that the edge is in collision and should be removed from the PRM.

the obstacles in the environment, converts them to depth pixels, encodes them in binary, and sends them sequentially to the processor. Empty entries in the occupancy grid are not sent. Upon receiving a depth pixel as input, *all collision detection circuits run simultaneously, in parallel*, to determine whether or not their edge collides with that particular depth pixel. After all of the depth pixels have been streamed to the chip, the collision bits stored for each edge indicate whether or not that edge is in collision. The robot retrieves the collision bits from the chip, eliminates the corresponding edges from the PRM, and then finds the least-cost path from start to goal using those that remain (Figure 3f.)

If no path is found to the goal configuration, then either obstacles have bisected the precomputed roadmap such that there is no collision-free path to the goal, or edges necessary to reach the goal were never present in the roadmap. In either case, the system can fall back on a conventional software planning routine that maintains probabilistic completeness. In this way, the common case would be drastically accelerated, while retaining the theoretical guarantees that make sampling algorithms attractive.

Note that the first four steps, while computationally intensive, need only happen once—after that, the specialized circuitry executes in parallel at very high speeds (our FPGA is clocked such that it takes less than 50 nanoseconds to resolve all collisions for a given depth pixel). Our process therefore performs one very expensive design-time computation that eliminates computational dependence on the complexity of the robot model—both its shape and degrees of freedom—and compiles the results into a structure that enables extremely fast run-time computation.

## IV. Subsampling the PRM

There are two major competing constraints on our design: first, we must use a single PRM, constructed in advance; second, we have only a finite amount of hardware at our disposal to construct circuits. The second constraint is a hard one, and usually takes the form of an edge budget: we can fit $n$ edges on our chip. Consequently, we would like to find the $n$-edge PRM most likely to be able to solve future motion planning problems.

We model this problem as a probabilistic one. In addition to accepting a description of the robot, we also use a *scenario distribution*: a distribution over problems (start state, end goal, and obstacles). The robot is expected to have to solve problems drawn from the scenario distribution, and the distribution itself expresses the structure inherent in the problem. A scenario distribution is often reasonably easy to describe for real-life applications; for example, any fixed obstacles are present with probability 1, and we expect the robot broadly to have goals that are in front of it and in a certain height range, rather than immediately behind its shoulder joint.

We must therefore optimize the PRM to maximize the probability of solving problems drawn from the scenario distribution. Approaches to compress PRMs exist (for example using graph spanners [7, 29, 31]), but to the best of our knowledge none are probabilistic in the sense that we require. Spanners attempt to maintain coverage properties, whereas we can use information about the expected probabilistic distribution of goals and obstacles to sacrifice coverage in uninteresting areas, while maintaining critical edges. We therefore adopt a simple heuristic approach: we draw many problems from the scenario distribution, build a very large initial PRM, and then prune it according to edge usage frequency in the sampled problems.

## V. Processor Design

### A. Overview

The processor consists of $N$ collision detection circuits (CDCs), each of which corresponds to one edge in the PRM. The primary input to the processor is a stream of depth pixels, which are fed in one at a time. Each pixel input is a $k$-bit entity (our current implementation uses 15 bits per pixel), and

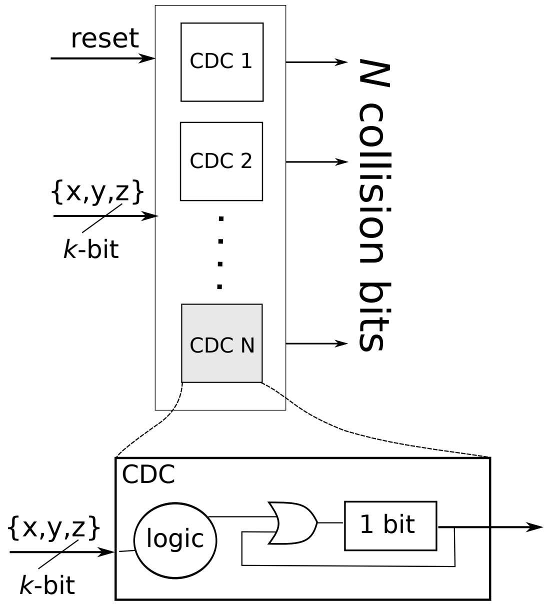


Fig. 4: The high level processor design features $N$ collision detection circuits, one for each edge. Each CDC contains edge collision circuitry plus a single bit of memory which stores whether or not any pixels have caused a collision since the last processor reset.

this input is fanned out to all of the CDCs. The output of the processor is an $N$-bit string (i.e., one bit per CDC), in which each bit corresponds to whether that edge collides with any of the depth pixels that have been streamed in since the most recent reset of the processor. We illustrate this high-level design in Figure 4. The processor also has secondary inputs (e.g., processor reset) for control purposes.

As each depth pixel input arrives at the chip, all of the CDCs operate simultaneously, in parallel, on that pixel. Thus, the latency to perform collision detection is linear in the number of depth pixels but independent of the number of edges in the PRM. This independence on the size of the PRM is the key advantage enabled by implementing the collision detection functionality in hardware.

### B. *Constructing a Collision Detection Circuit*

We now explain in detail how we construct a collision detection circuit for a specific edge, using the running example of a simple 2D robot in a workspace discretized to an $8 \times 8$ grid. Each depth pixel is represented by a pair of 3-bit numbers, one each for $x$ and $y$; we denote their binary digits as $x_1, x_2, x_3$ and $y_1, y_2, y_3$.

Consider edge $i$, which corresponds to a swept volume that collides with just two depth pixels, at locations $(6, 1)$ and $(5, 1)$ (which have binary representations $(110, 001)$ and $(101, 001)$, respectively). Given a depth pixel described as $d = (x_1, x_2, x_3, y_1, y_2, y_3)$, the logical expression that determines whether it collides with this specific edge is:

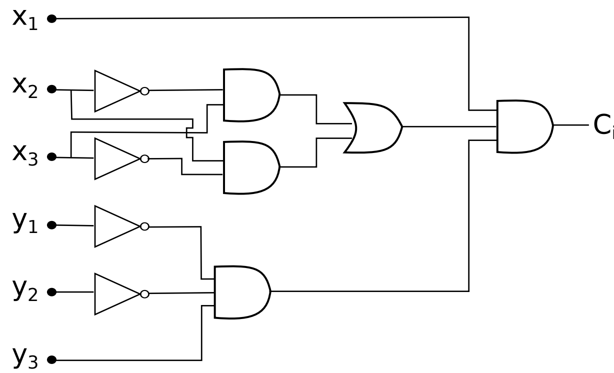


Fig. 5: The circuit representing the logical expression given in Equation 1.

$$C_i = (x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg y_1 \wedge \neg y_2 \wedge y_3) \vee (x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg y_1 \wedge \neg y_2 \wedge y_3).$$

The above expression has one conjunction per depth pixel, but can be simplified to:

$$C_i = (\neg y_1 \wedge \neg y_2 \wedge y_3) \wedge x_1 \wedge ((\neg x_2 \wedge x_3) \vee (x_2 \wedge \neg x_3)), \quad (1)$$

as the $y$-coordinates in this case are the same, and $x_1$ is true in both cases. This results in the Boolean circuit in Figure 5.

In addition to optimizing each circuit in isolation, we also have the opportunity to optimize *across* the individual collision circuits. We expect that many swept volumes will overlap and have depth pixels in common; the circuitry for these common expressions can be shared, saving hardware and ultimately allowing us to use a larger PRM.

Each CDC includes one bit of storage to record whether that CDC has detected a collision with any depth pixel input since the most recent reset of the processor. A reset of the processor causes this bit to be cleared in every CDC.

### C. *Implementation*

We specified the circuitry using the Verilog hardware description language and used CAD tools to compile the circuitry into a format which can be downloaded onto a field programmable gate array (FPGA). An FPGA is a chip consisting of digital logic that can be reconfigured to provide the functionality of a downloaded circuit. FPGAs thus provide great flexibility, particularly for prototyping efforts.

## VI. EXPERIMENTS

We now describe the results of applying our approach to two planning problems—one in which a robot arm must perform pick-and-place in an environment with obstacles, and another in which it must reach a target in the presence of vertical poles. Our experiments made extensive use of the Klampt robot simulator [9], and the `kinect2_bridge` [32] and `ar_track_alvar` [21] ROS packages.

## A. Pick-and-Place

Our first experiment involves a pick-and-place scenario where a 6-degree-of-freedom Kinova Jaco-2 arm must move colored blocks from a table to one of two bins. The blocks, as well as some boxes (which act as obstacles), are placed uniformly at random on the table. Figure 6 shows a real instance of this problem, as well as three randomly generated sample environments.

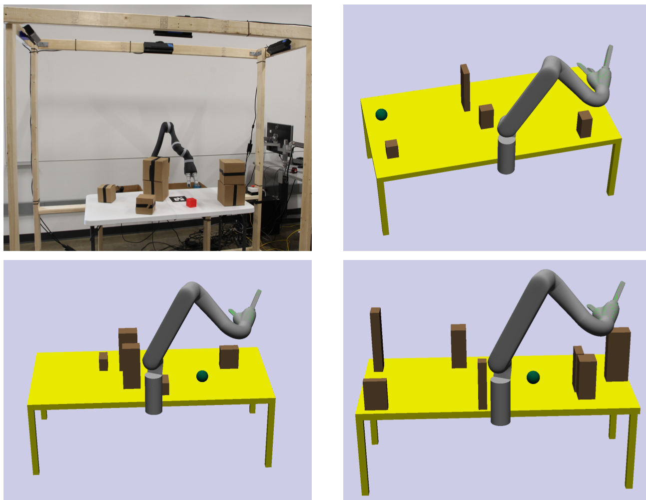


Fig. 6: A real instance of our pick-and-place task, as well as three randomly created environments simulated using Klampt.

Four Kinect-2 sensors mounted above the table on a wooden frame were used for sensing. Each Kinect generates a point cloud; the four point clouds were initially registered via an AR Tag [12] placed on the empty table, and then merged and converted into an occupancy grid at runtime using a simple probabilistic filter. An example obstacle configuration and occupancy grid are shown in Figure 7.

The planner's goal was to find a grasp pose where the gripper was within 10cm of the center of the target and pointing downward. This enabled us to use a Cartesian controller to actually perform the grasp. In addition, the planner had to be able to find a path to a pose 10cm above one of the two bins mounted on the back of the frame, at which time we opened the gripper to drop the target object off.

We built a $150,000$ edge PRM for this problem using Klampt's implementation of PRM*, and used the pruning method described in Section IV with $10,000$ sample environments to reduce its size. Each sampled environment

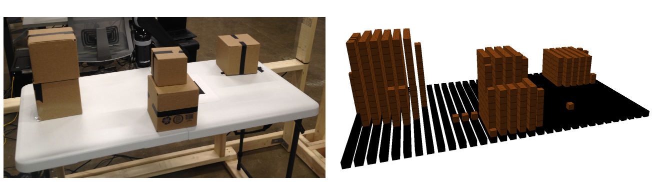


Fig. 7: A configuration of obstacles (left) and the corresponding sensed occupancy grid (right).

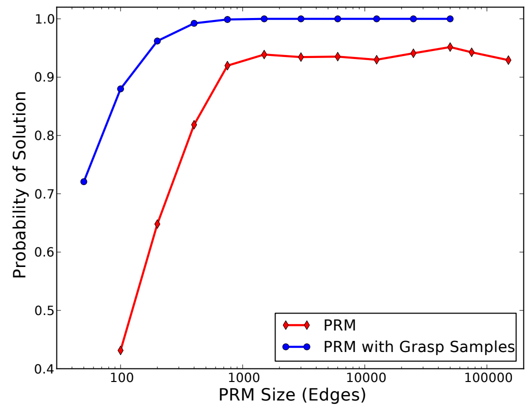



Fig. 8: The probability of finding a plan when pruning a PRM (red) or a PRM with extra grasp samples (blue).

included the same fixed "environment obstacle" (the table), but contained different numbers of randomly located obstacles of variable size. In this way the robot was faced with some easy scenarios and some quite challenging ones, and the frequency of edge usage could be tracked; this enabled our heuristic pruning. We then tested the ability of the smaller pruned PRM to solve a new set of $10,000$ sample environments, and iterated in this manner (pruning and retesting) to see how many edges we actually needed. The results indicated that we could reduce the PRM to approximately $10,000$ edges without dropping the probability of finding a solution, but they also showed that the initial large PRM was inadequate, only capable of solving $90\%$ of problems even at full size. Our analysis of the failure cases indicated that insufficient sampling at grasping height was to blame, which led to an inability to find a suitable goal node. To address this we added $100$ grasping poses to a freshly generated large PRM ($50,000$ edges). This PRM could be subsampled down to fewer than $1,000$ edges while still solving all sampled problems. These results are shown in Figure 8.

We therefore used a $1024$-edge PRM (the FPGA can fit up to 2500 edges) to construct collision detection circuitry for this problem. We implemented the circuitry on a Stratix V FPGA on Terasic's DE5 development board, interfaced to a desktop computer using PCIe. We first used Klampt to exhaustively compute all the points in the discretized space above the table that collided with each edge. The binary representations of the points were used to construct truth tables representing the logic functions of groups of $16$ edges.

The next step was to optimize the logic functions to reduce their size. Logic minimization is a well-studied problem due to its usage in electronic design automation tools. We used a version of the popular *espresso* heuristic logic minimizer [6, 28]. This tool allows for group minimization across $16$ outputs at a time. The resulting functions were then converted into Verilog. Using *espresso* prior to converting to Verilog enabled greater than $25\%$ savings in logic utilization on the

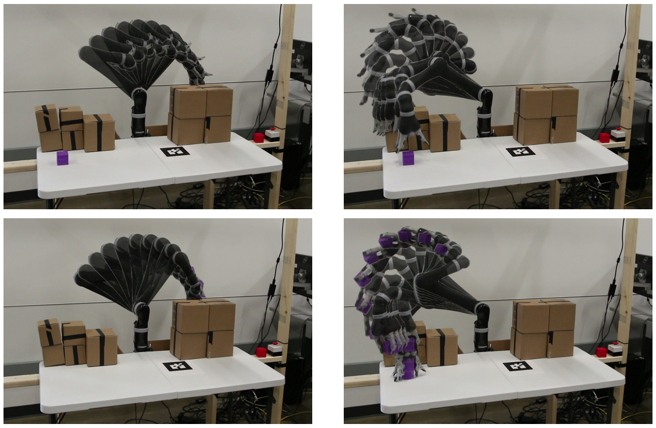


Fig. 9: An example planned movement from the pick-and-place domain, visualized clockwise from top left. The Jaco arm navigates around the obstacle stack on the left to pick up the purple block and drop it in the collection box.

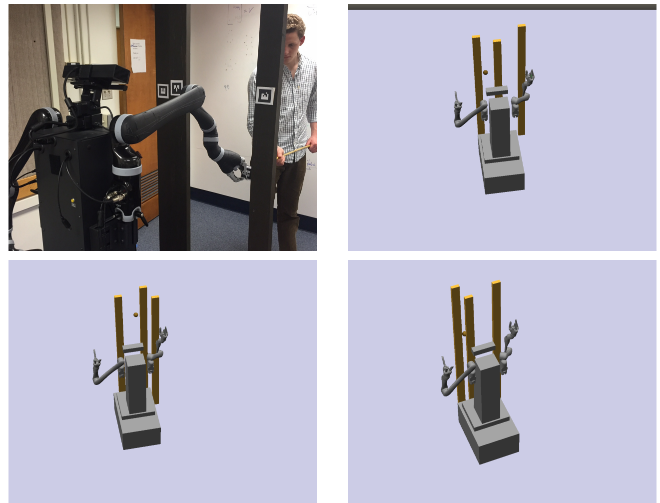


Fig. 10: A real instance of our vertical obstacle task, as well as three randomly created environments simulated using Klampt.

FPGA, even though all FPGA tools attempt to do some logic minimization on their own. It is likely that we could realize even more hardware savings using more carefully selected edge groupings prior to optimization.

We measured the total time this implementation uses for motion planning (starting with an occupancy grid, and hence excluding the cost of transforming video or depth-cloud data into an occupancy grid) averaged over $1,000$ randomly drawn environments, and compared it to various software implementations. We used the software versions of PRM and RRT in Klampt; details and source code are available [9]. The resulting timing data are shown in Table I. Our method is able to solve such problems in less than a single millisecond on average—the vast majority of which is spent communicating with the FPGA and performing a graph search, which together account for $87\%$ of planning time. This represents the removal of the collision detection bottleneck, shifting that distinction onto other aspects of motion planning.

We tested the circuitry on many configurations of the real instantiation of the pick-and-place task. We designed the system so that it blocked until the user pressed a red button (visible on the right of the wooden frame), whereupon planning was allowed to commence. Planning was instantaneous, and the robot started moving immediately after the button press without any noticeable delay. One solution is visualized in Figure 9.

### *B. Vertical Obstacles*

Our second experiment has a mobile robot, Anathema Device, placed in front of three movable vertical obstacles and asked to reach a target without colliding with any of them. Three randomly sampled configurations (simulated in Klampt) and the physical arrangement are shown in Figure 10.

In this case we used the Kinect-2 on board Anathema to identify AR Tags attached to each vertical obstacle, which were in turn used to build the occupancy grid based on our knowledge of the size of the vertical obstacles. We also used an AR Tag to identify the goal, which was held in front of the robot between waist and chest height using a long handle.

We used Klampt to build a large PRM—$25,000$ edges—-and a similar procedure to subsample down to $825$ edges, which were converted into collision circuits and realized on the same FPGA. Timing results are shown in Table I, again showing that our implementation finds paths for this tricky problem in less than one millisecond. One solution is visualized in Figure 11.

## VII. Related Work

The closest related work is the design of hardware, intended for an FPGA, to accelerate the PRM algorithm directly [1]. The parallelism in this method is largely drawn from implementing multiple triangle intersection functional units. This speeds up the testing of whether or not the mesh models of the robot and obstacles collide, by performing multiple pair-

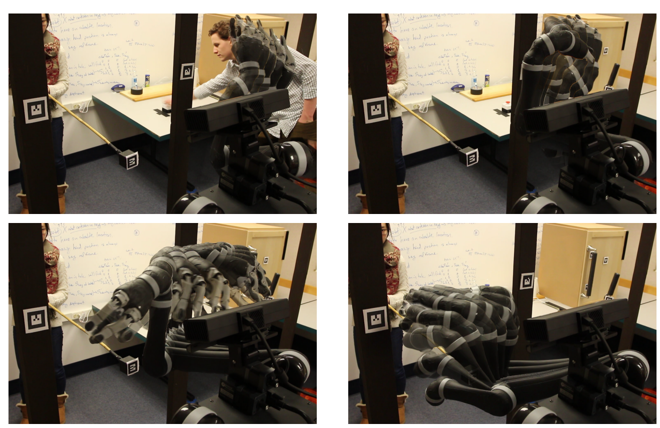


Fig. 11: An example planned movement from the vertical obstacle domain, visualized clockwise from top left.

| Environment | FPGA-Accelerated Planning | | | | | | Software | |
|---|---|---|---|---|---|---|---|---|
| | Goal ID | Comm. | Collision | Del. Edges | Path | Total | PRM | RRT |
| Pick-and-Place | 13 | 118 | 16 | 50 | 425 | **622** | 815,000 | 756,000 |
| Vertical Obstacles | 55 | 104 | 16 | 32 | 420 | **627** | 2,738,000 | 1,074,000 |

TABLE I: The time (in microseconds) required to perform path planning on the proposed system, broken into its components: identifying goal nodes in the PRM, communication over PCIe, collision detection, deleting edges that are in collision from the PRM, and performing path search. We have included reference times for two representative software planners available in Klampt; these were run on a 4-core Intel Xeon processor clocked at 3.5 GHz with 16 GB of RAM.

wise checks in parallel. Unfortunately, triangle intersection testing involves a very large amount of arithmetic (especially multiplications), and even high-capacity FPGAs cannot fit nearly enough multipliers to make this design feasible. Consequently the authors were unable to fit their design on an FPGA for their somewhat simplified test case. Furthermore, even had their design been feasible, it would have provided only a constant-factor speedup—parallelizing a small number of triange-triangle tests—whereas our CDCs all execute in parallel and thus have constant run-time complexity with respect to roadmap size.

There is some additional related prior work in using special-purpose processors to accelerate collision detection for applications other than motion planning [27, 34, 35]. These processors are designed to accelerate graphics and physical simulations. As such, these prior designs have very different requirements. The graphics and simulation applications must determine not just whether two moving objects collide, but when and where they collide, with a high degree of precision. For robotics, we simply need to avoid collisions, and it is acceptable to be conservative. Because of our simpler requirements, we can sacrifice precision to achieve real-time performance.

## VIII. Discussion and Conclusions

The major constraint on our implementation is the amount of hardware available to represent PRM edges. Although our results are very promising, they are on relatively small PRMs, which consumed a substantial portion of the capacity of our high-capacity FPGA. However, a key attribute of our approach is that the time to perform collision detection is independent of the number of edges in the PRM—which means that it is, by virtue of its parallelism, algorithmically faster (by a factor linear in the number of edges) than any prior software scheme. Consequently, other hardware platforms that are better suited to our specific design could plan using much larger PRMs at roughly the same speed. For example, our FPGA implementation, while flexible, is a prototype device that sacrifices the capacity, performance, and power-efficiency that could be achieved with an application specific integrated circuit (ASIC). Our rough calculations* using industry-standard CAD software from Synopsys indicate that $1,000$ collision detection circuits could be implemented on an ASIC using fewer than 10 million transistors. Since even relatively old CMOS processes can fit 1 billion transistors on a single chip, we predict that we could comfortably fit $100,000$ edges on an ASIC. This would allow the chip to use roadmaps of sufficient complexity to solve the real planning challenges present in many industries. There is an inherent trade-off between the low initial investment and high flexibility of an FPGA, and the high once-off cost but low unit cost of an ASIC, which can also yield higher capacity, performance, and power-efficiency.

There is also a trade-off when determining the resolution of discretization between more accurate representations of the environment (which is important in low-clearance situations), and the amount of resources on the FPGA each edge consumes. Doubling the resolution in all dimensions leads to a roughly 4x increase in hardware usage (but has no significant effect on execution time).

Another limitation of our approach is that the PRM must be constructed in advance. Consequently, if the robot's physical configuration changes (for example, an arm is bent) the planning circuitry may no longer be accurate. Of more concern is that the same problem applies if the robot picks up an object that substantially exceeds the size of its gripper. However, since reconfiguring the FPGA takes just a few seconds, an FPGA-based implementation could adjust the PRM in software (e.g., in response to a bent arm), or carry a large number of PRMs—perhaps one each for a finite number of grasped object bounding boxes—and swap them out at runtime. Any delay could be avoided by switching between two FPGAs, one of which is re-programmed during plan execution. In future work, we also plan to investigate adding extra bits that can activate one of a small number of grasped object bounding boxes for each swept volume.

Designing specialized hardware is an unusual step. However, the motion planning problem has two important characteristics which make building specialized hardware for it worthwhile: first, that it is basic to robot movement, and therefore critical to do quickly and well; and second, the problem affords massive parallelism which can be exploited in hardware. Our results show that such an implementation is indeed feasible, and provides performance that is several orders of magnitude faster than the previous planners. Moreover, although our experiments have demonstrated only kinematic planning, our approach is applicable to any problem amenable

*We multiplied the number of transistors for each gate type by the number of that type in the design (e.g., 6 transistors for 2-input AND and 2-input OR). Because we configured the software to use "medium effort", these results are likely somewhat pessimistic.

to a roadmap-style approach. The circuitry described here performs motion planning faster than perception using off-the-shelf sensors like the Kinect, which operates at 30Hz, and opens up new possibilities beyond simple once-off planning. Motion planning could be done in real-time in an environment with moving obstacles and/or goals, or run in parallel with the current motion. It is now even fast enough to be used as a subroutine of some other software component—for example, as is often required in combined task and motion planning algorithms [33, 10, 30, 5]. These examples illustrate the core role that motion planning will play in designing generally capable robots, once it is fast enough. Specialized motion planning hardware is a promising approach to solving this key problem.


## Acknowledgments

We owe a great deal of thanks to Kris Hauser for developing the Klampt robotics package [9], and for his assistance in becoming familiar with the software. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) Robotics Fast Track Program. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of DARPA OSRF, or the US government. We are grateful to Altera Corporation for their support on this project. We would additionally like to thank Xiangyu Zhang, Barrett Ames, Saeed Alrahama, Hayden Bader, and Martha Barker.


## References


[1] N. Atay and B. Bayazit. A motion planning processor on reconfigurable hardware. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 125–132, 2006.

[2] K. E. Bekris and L. E. Kavraki. Greedy but safe replanning under kinodynamic constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 704–710, April 2007.

[3] J. Bialkowski, S. Karaman, and E. Frazzoli. Massively parallelizing the RRT and the RRT$^*$. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3513–3518, 2011.

[4] J. Bialkowski, S. Karaman, M. Otte, and E. Frazzoli. Efficient collision checking in sampling-based motion planning. In *Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics*, pages 365–380, 2013.

[5] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti. Geometric backtracking for combined task and motion planning in robotic systems. *Artificial Intelligence*, 2015.

[6] R. Brayton, G. Hatchel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Boston, MA, 1984.

[7] A. Dobson and K. E. Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. *International Journal for Robotics Research*, 33(1):18–47, 2014.

[8] N. K. Govindaraju, S. Redon, M.C. Lin, and D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, page 2532, 2003.

[9] K. Hauser. Robust contact generation for robot simulation with unstructured meshes. In *Proceedings of the International Symposium on Robotics Research*, 2013.

[10] L. Kaelbling and T. Lozano-Pérez. Hierarchical planning in the Now. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1470–1477, 2011.

[11] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[12] H. Kato and M. Billinghurst. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, 1999.

[13] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[14] D. Knott and D. K. Pai. CInDeR: Collision and interference detection in real-time using graphics hardware. In *Proceedings of Graphics Interface*, pages 73–80, 2003.

[15] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[16] S. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 23(5):378–400, 2001.

[17] J. Lengyel, M. Reichert, B.R. Donald, and D.P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, pages 327–335, 1990.

[18] P. Leven and S. Hutchinson. A framework for real-time path planning in changing environments. *The International Journal of Robotics Research*, 21(12):999–1030, 2002.

[19] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C32(2): 108–120, 1983.

[20] K. Myszkowski, O. G. Okunev, and T. L. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9): 497–511, 1995.

[21] S. Niekum. ar_track_alvar. http://wiki.ros.org/ar_track_alvar, 2011 – 2015.

[22] Michael Otte and Emilio Frazzoli. RRT-X: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research*, 2015.

[23] J. Pan and D. Manocha. GPU-based parallel collision detection for fast motion planning. *International Journal of Robotics Research*, 31(2):187–200, 2012.

[24] J. Pan, C. Lauterbach, and D. Manocha. g-Planner: Real-time motion planning and global navigation using GPUs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010.

[25] J. Pan, C. Lauterbach, and D. Manocha. Efficient nearest-neighbor computation for GPU-based motion planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2243–2248, 2010.

[26] C. Park, J. Pan, and D. Manocha. Real-time optimization-based planning in dynamic environments using GPUs. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4090–4097, 2013.

[27] A. Raabe, S. Hochgurtel, J. Anlauf, and G. Zachmann. Space-efficient FPGA-accelerated collision detection for virtual prototyping. In *Proceedings of the Design Automation Test in Europe Conference*, pages 206–211, March 2006.

[28] R.L. Rudell. Multiple-valued logic minimization for pla synthesis. Technical Report UCB/ERL M86/65, EECS Department, University of California, Berkeley, 1986.

[29] O. Salzman, D. Shaharabani, P. K. Agarwal, and D. Halperin. Sparsification of motion-planning roadmaps by edge contraction. *The International Journal of Robotics Research*, 33(4):1711–1725, 2014.

[30] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 639–646, 2014.

[31] W. Wang, D. Balkcom, and A. Chakrabarti. A fast streaming spanner algorithm for incrementally constructing sparse roadmaps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1257–1263, 2013.

[32] T. Wiedemeyer. IAI Kinect2. https://github.com/code-iai/iai_kinect2, 2014 – 2015. Accessed June 12, 2015.

[33] J. Wolfe, B. Marthi, and S. Russell. Combined Task and Motion Planning for Mobile Manipulation. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, pages 254–258, 2010.

[34] M. Woulfe, J. Dingliana, and M. Manzke. Hardware Accelerated Broad Phase Collision Detection for Realtime Simulations. In *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS"*. The Eurographics Association, 2007.

[35] G. Zachmann and G. Knittel. An architecture for hierarchical collision detection. In *Journal of WSCG*, volume 11, pages 149–156. February 2003.